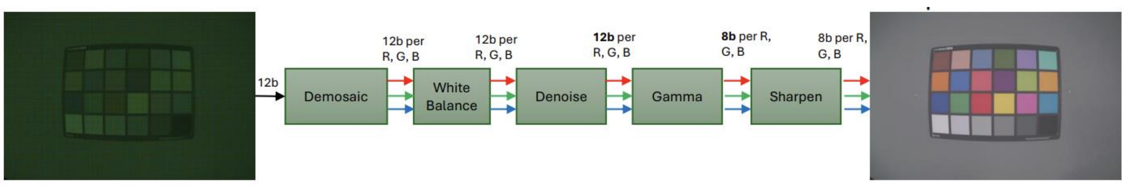
Emmetra Assignment -1

Problem Statement :

– Implement basic image signal processing routines for sensor raw image

Tasks :

- Implement following routines with option to control the parameters of the algorithms
- Demosaic – edge based interpolation (5x5) to compute missing channels
- White balance – simple gray world algorithm to remove color cast.
- Denoise – Gaussian filter (5x5)
- Gamma correction – use sRGB gamma (convert 12b to 8b)
- Sharpening filter – unsharp mask filter



My Approach

- **Setup & Configuration**: Initialized a Flask app with settings for secure file uploads and a cache for storing processing parameters and results.

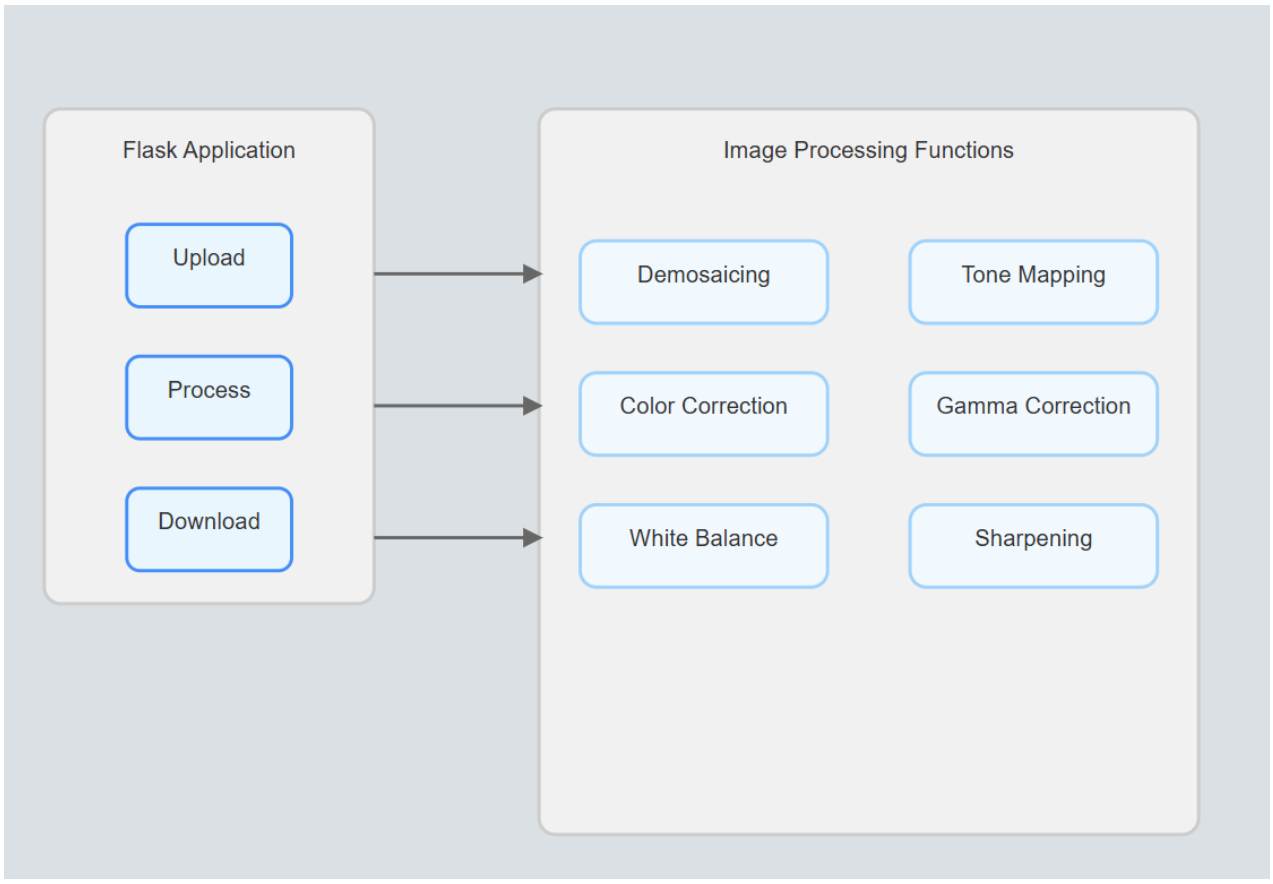
- **File Upload Handling**: Allows users to upload RAW Bayer images, which are stored with unique timestamps for secure management.

- **Image Processing Pipeline**: Processes images using a series of ISP functions—demosaicing, tone adjustment, white balance, color correction, tone mapping, and gamma correction. Optional steps include denoising, sharpening, and Gaussian blurring based on user-specified parameters.

- **Parameter Adjustment**: Allows users to adjust processing parameters via HTTP requests, updating the cached values to customize the pipeline's behavior per image.

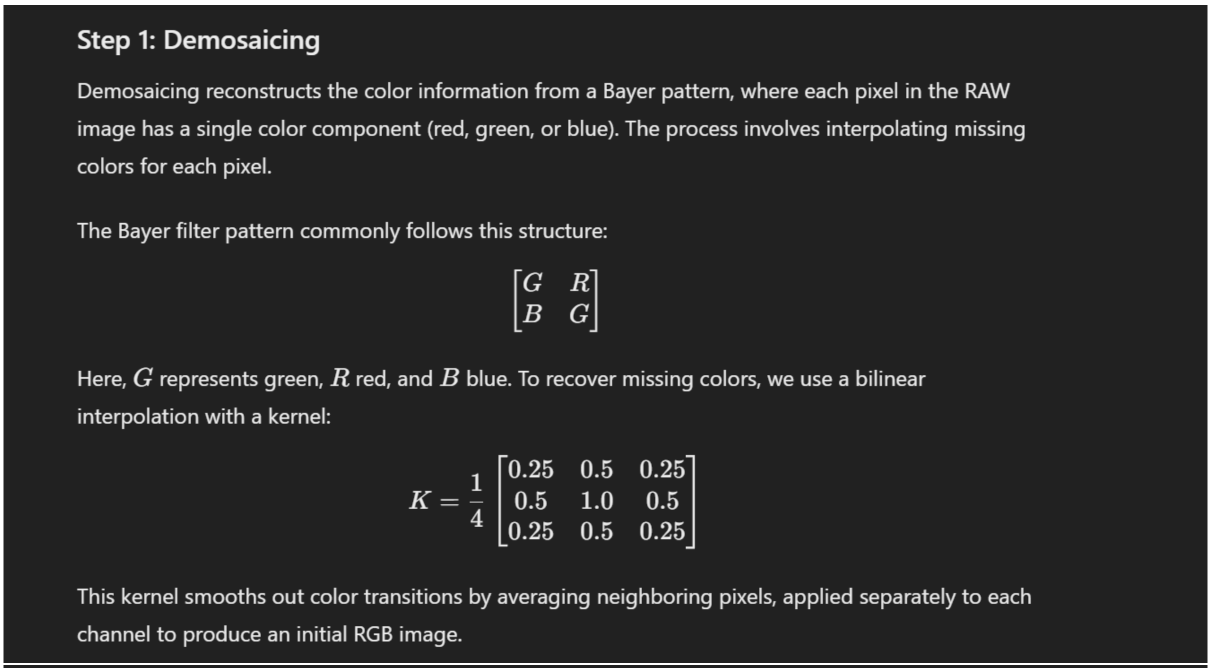
- **Download Processed Images**: Saves the final processed image in PNG format for download, and includes error handling for file size limits and server errors.

System design diagram



ISP Pipeline explained

Here is how the ISP pipeline is working-



Step 1: Demosaicing

Demosaicing reconstructs the color information from a Bayer pattern, where each pixel in the RAW image has a single color component (red, green, or blue). The process involves interpolating missing colors for each pixel.

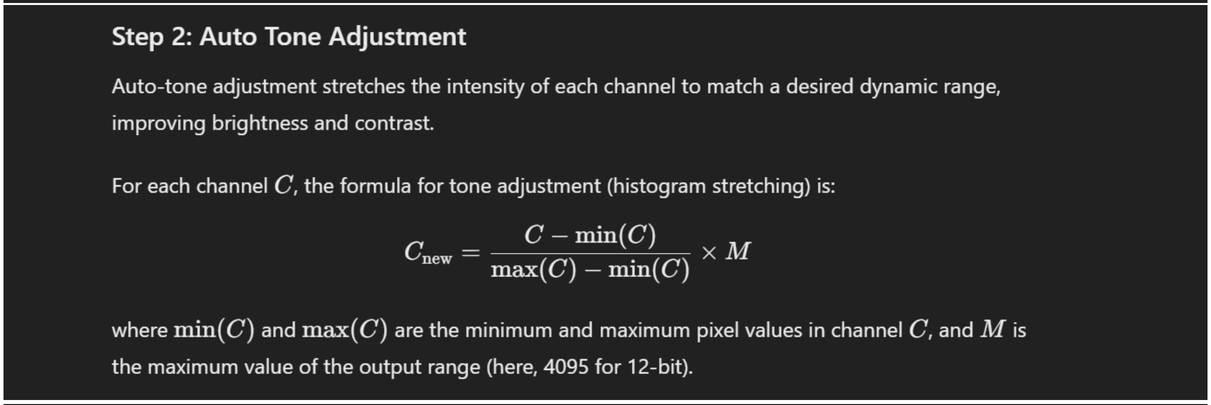
The Bayer filter pattern commonly follows this structure:

$$\begin{bmatrix} G & R \\ B & G \end{bmatrix}$$

Here, G represents green, R red, and B blue. To recover missing colors, we use a bilinear interpolation with a kernel:

$$K = \frac{1}{4} \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1.0 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}$$

This kernel smooths out color transitions by averaging neighboring pixels, applied separately to each channel to produce an initial RGB image.



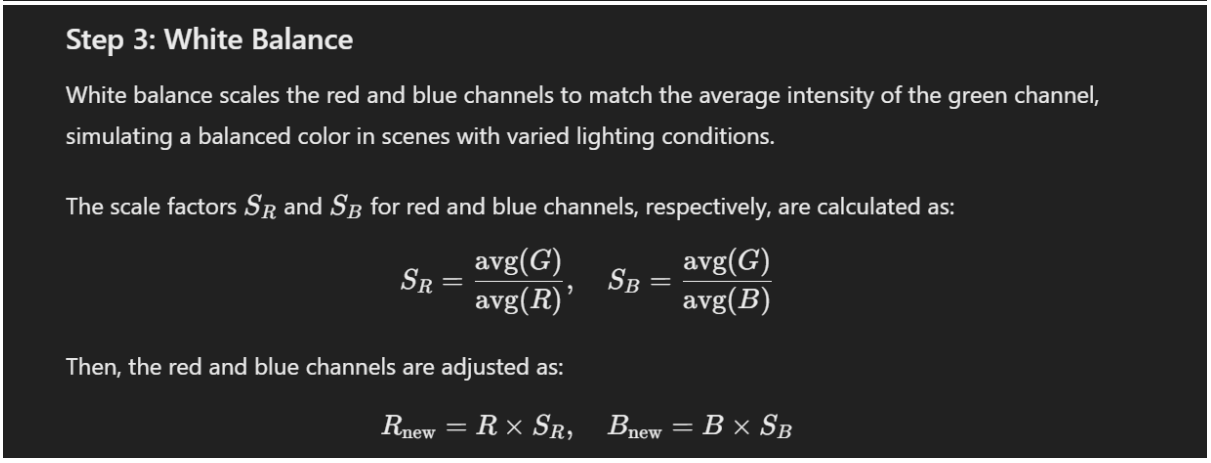
Step 2: Auto Tone Adjustment

Auto-tone adjustment stretches the intensity of each channel to match a desired dynamic range, improving brightness and contrast.

For each channel C, the formula for tone adjustment (histogram stretching) is:

$$C_{\text{new}} = \frac{C - \min(C)}{\max(C) - \min(C)} \times M$$

where $\min(C)$ and $\max(C)$ are the minimum and maximum pixel values in channel C, and M is the maximum value of the output range (here, 4095 for 12-bit).



Step 3: White Balance

White balance scales the red and blue channels to match the average intensity of the green channel, simulating a balanced color in scenes with varied lighting conditions.

The scale factors S_R and S_B for red and blue channels, respectively, are calculated as:

$$S_R = \frac{\text{avg}(G)}{\text{avg}(R)}, \quad S_B = \frac{\text{avg}(G)}{\text{avg}(B)}$$

Then, the red and blue channels are adjusted as:

$$R_{\text{new}} = R \times S_R, \quad B_{\text{new}} = B \times S_B$$

Step 4: Color Correction

Color correction enhances color balance and vividness by applying a color correction matrix CCM to each pixel. For a pixel with RGB values $\vec{p} = [R, G, B]^T$, we have:

$$\vec{p}_{\text{corrected}} = CCM \cdot \vec{p}$$

The color correction matrix in this code is:

$$CCM = \begin{bmatrix} 1.5 & -0.2 & -0.2 \\ -0.2 & 1.5 & -0.2 \\ -0.2 & -0.2 & 1.5 \end{bmatrix}$$

This boosts the diagonal elements (primary colors) while reducing cross-channel contamination to make colors more vivid.

Step 5: Tone Mapping and Gamma Correction

Tone mapping compresses the dynamic range to fit within the displayable range. Two tone mapping functions are provided here:

1. **Adaptive Logarithmic Tone Mapping**: Applies a logarithmic compression:

   $$I_{\text{mapped}} = \frac{\log(1 + k \cdot I)}{\log(1 + k)}$$

   where k controls the compression, and I is the normalized intensity.

2. **Adaptive Power-Based Tone Mapping**: Compresses intensity using a power function:

   $$I_{\text{mapped}} = I^{\text{compression factor}}$$

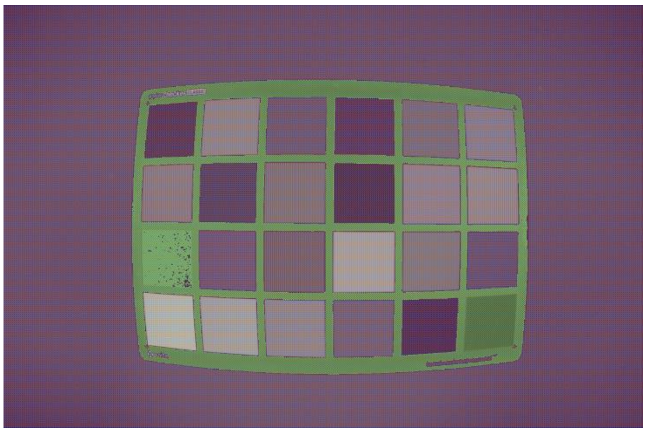
   A lower compression factor preserves contrast in darker regions.

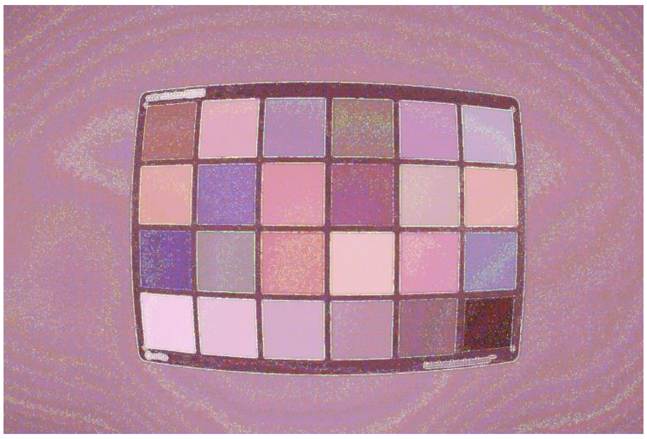
Finally, **Gamma Correction** corrects the intensity nonlinearly for display, adjusting each channel C as:

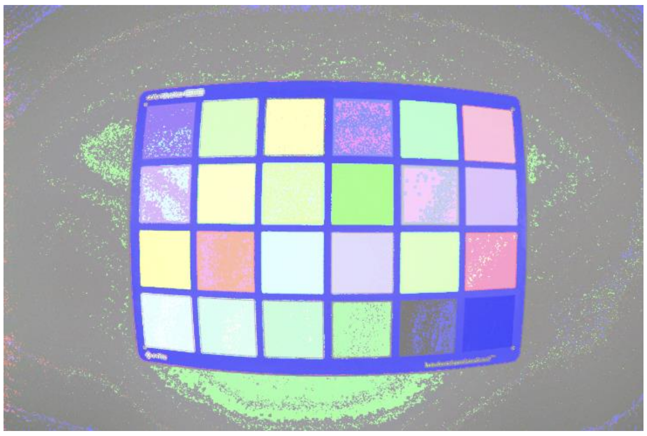
$$C_{\text{final}} = \left(\frac{C}{255}\right)^{1/\gamma} \cdot 255$$

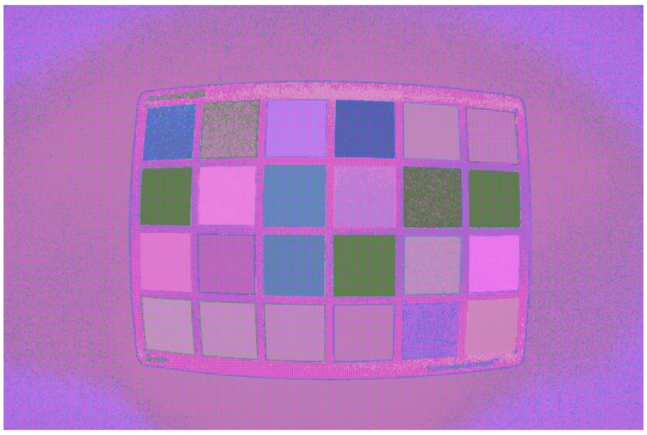
with a typical gamma value of 2.2, converting the image to an 8-bit format for display.

Initial Outputs-



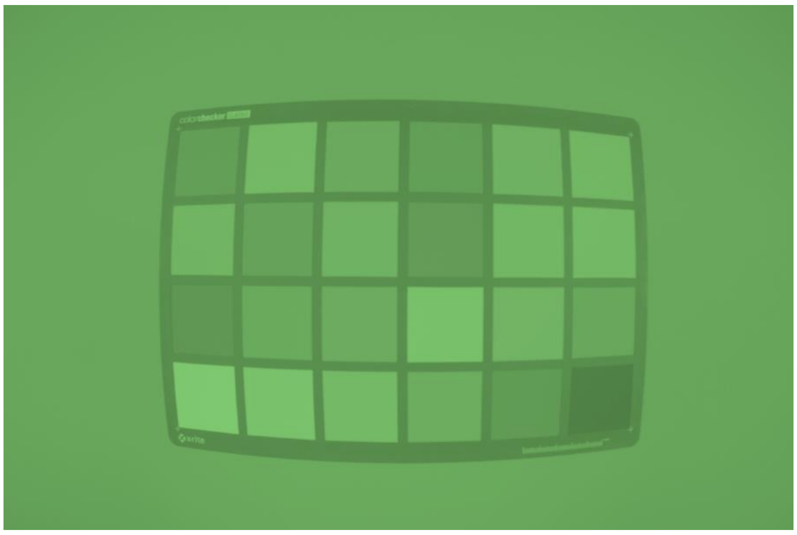




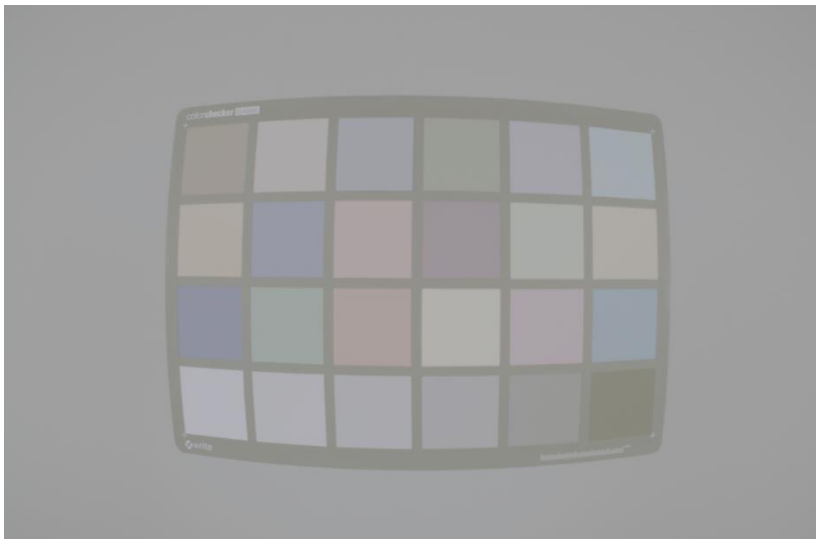


Reason – Inefficient Demosaicing algorithm

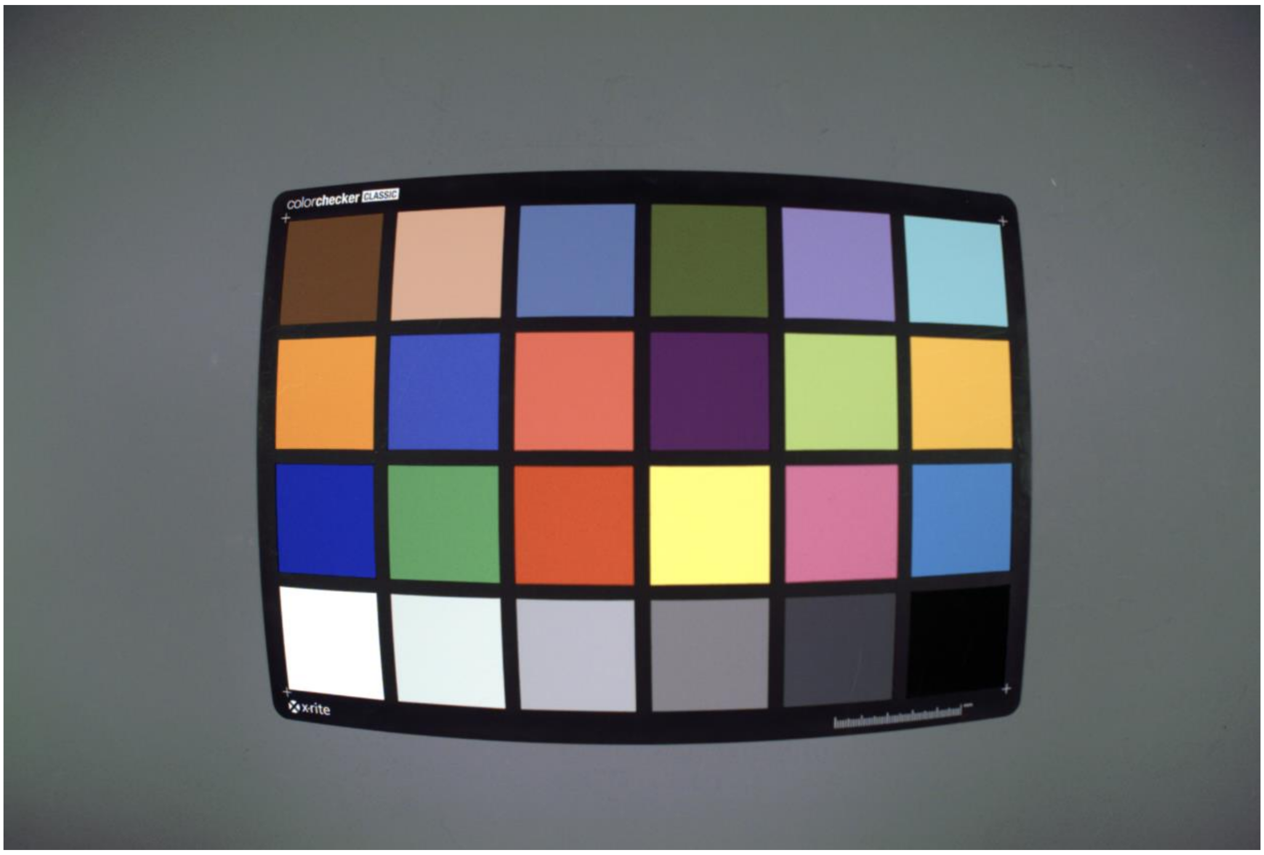
After Correction -



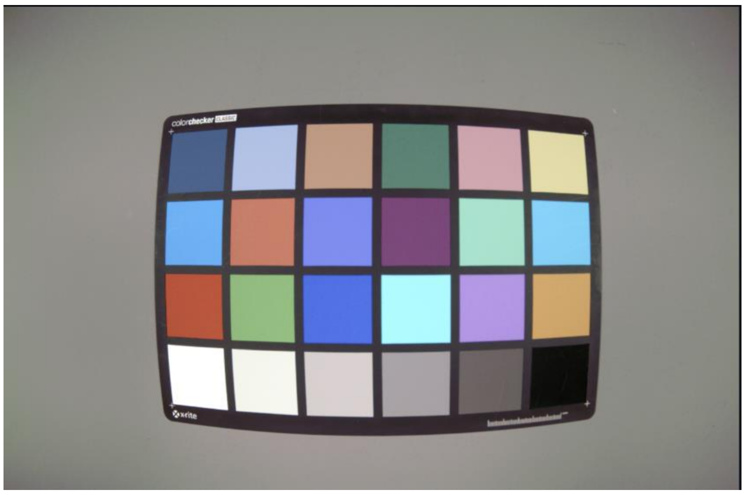
After applying proper color matrix and non linear tone mapping

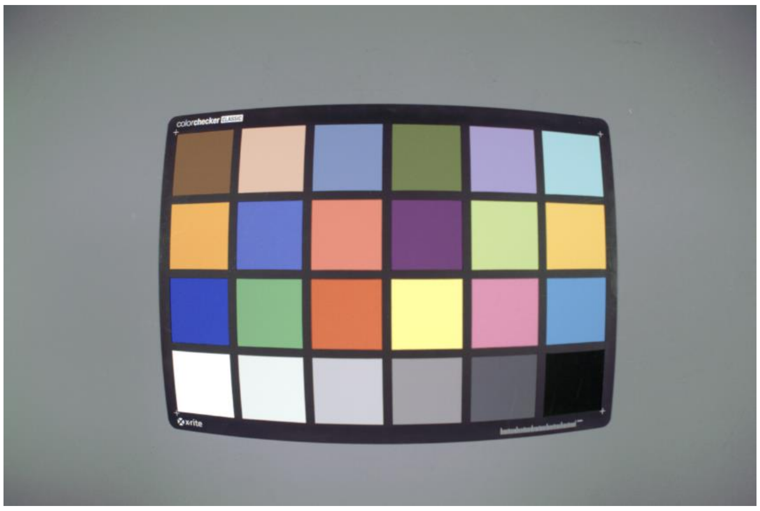


After correcting Gamma , applying auto tone algorithm and compression algorithms –



Swapping red and blue spaces –

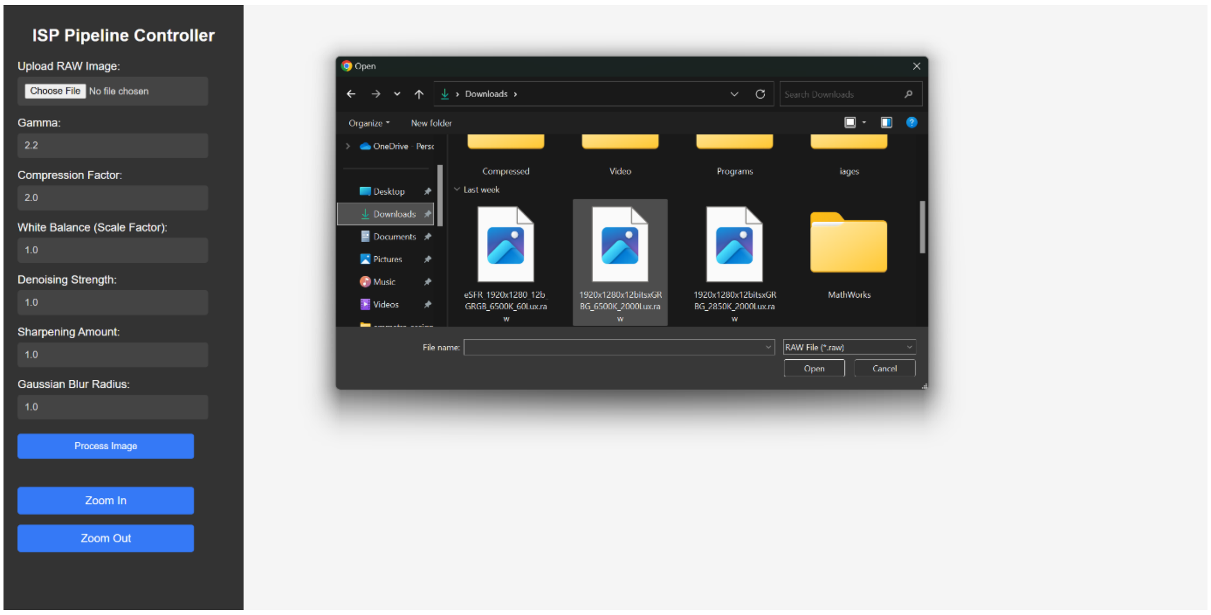




Adjusted according to pixelviewer's demosaicing.

Website Outputs –

Chose raw file


ISP Pipeline Controller

Upload RAW Image: No file chosen

Gamma: 2.2

Compression Factor: 2.0

White Balance (Scale Factor): 1.0

Denoising Strength: 1.0

Sharpening Amount: 1.0

Gaussian Blur Radius: 1.0

Open

Downloads

Compressed | Video | Programs | iages

Last week

eSFR_1920x1280_12b_GRGB_6500K_60Lux.raw

1920x1280x12bitsxGRBG_6500K_2000Lux.raw

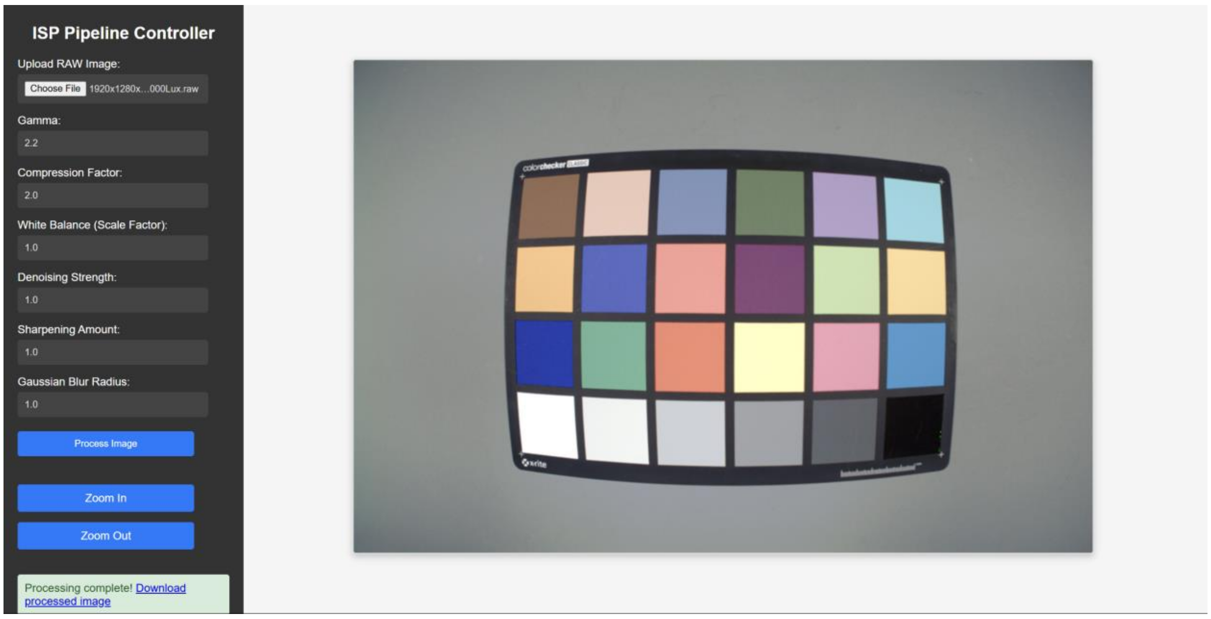
1920x1280x12bitsxGRBG_2850K_2000Lux.raw

MathWorks

File name:

RAW File (*.raw)

Image processed


ISP Pipeline Controller

Upload RAW Image: 1920x1280x...000Lux.raw

Gamma: 2.2

Compression Factor: 2.0

White Balance (Scale Factor): 1.0

Denoising Strength: 1.0

Sharpening Amount: 1.0

Gaussian Blur Radius: 1.0

Processing complete! Download processed image

Then download the image converted from 12bit raw to 8bit rgb.

Experiments

At first we implemented OpenCV version, while it was working, we were not satisfied and decided to implement a custom ISP pipeline using custom functions and methods taught in the bootcamp.

Observations

- **Subtle Color Variations in White Balance:** Adjusting white balance manually provided insight into how different lighting conditions affect color balance. Even slight shifts in white balance had a noticeable impact, especially on skin tones and natural scenes, highlighting the importance of accurate calibration.

- **Complexity of Demosaicing Algorithms:** The demosaicing step revealed how intricate algorithms can be for reconstructing high-quality images from Bayer data. Gradient-based and bilinear interpolation both showed advantages in certain scenarios, with gradient-based methods preserving edges better and reducing color artifacts.

- **Dynamic Range and Tone Mapping:** Experimenting with different tone-mapping functions demonstrated how adaptive methods could compress the dynamic range effectively while preserving contrast. This was particularly evident in scenes with high brightness variation, where logarithmic tone mapping was beneficial for handling highlights without losing detail.